ServoCenter 4.1

Volume 4:

Sequencer / SC-BASIC Interpreter Language Reference & Programming Guide

Yost Engineering, Inc.
630 Second Street
Portsmouth, Ohio 45662

www.YostEngineering.com

©2002-2009 Yost Engineering, Inc.
Printed in USA

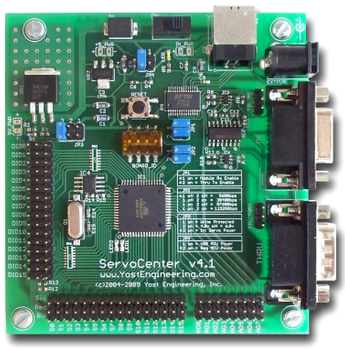
Table of Contents

ServoCenter 4.1 Sequencer / SC-BASIC Language Interpreter....4
1. Sequencer Overview....4
2. Sequencer Features....4
2. Language Reference....5
2.1 Comments....5
2.2 Variables....5
2.2.1 Variable Basics....5
2.2.2 Variable Scope and Lifetime....5
Global Variables....5
Procedure Variables....5
2.2.3 Variable Names....5
2.2.4 Declaring Variables....6
Implicit Declaration....6
Explicit Declaration....6
2.2.5 Using Variables....6
Variable Assignment....6
Variable Values....6
2.3 Arithmetic Operations and Expressions....7
2.3.1 Arithmetic Operations....7
2.3.2 Arithmetic Order of Operations....7
2.3.3 Arithmetic Operation Examples....7
2.3.4 Arithmetic Expressions....8
2.4 Relational and Logical Operations....8
2.4.1 Relational Operators....8
2.4.2 Logical Operators....8
2.4.3 Relational / Logical Order of Operations....9
2.4.4 Relational / Logical Operation Examples....9
2.5 Language Statements....9
2.5.1 Print Statement....9
Print Statement Details....9
Print Statement Examples....10
2.5.2 If Statements....10
Single Line If Statement Forms....10
2.5.3 For / Next Loops....11
For / Next Statement Forms....11
Exit For Statement....11
2.5.4 Do While Loops....11
Do While / Loop Forms....11
Exit While Statement....11
Infinite Loops....12
2.5.5 Procedures: Subs and Functions....12
Procedure Basics....12
Defining a Sub....12
Calling a Sub....12
Passing Information to a Sub....13
Exit Sub Statement....13
Defining a Function....13
Specifying Function Return Values....13
Calling a Function....14
Exit Function Statement....14
2.5.6 The End Statement....14
2.6 Built-in Functions....14
Pass( )....14
SetDIOHigh(DIONum)....14
SetDIOLow(DIONum)....14
SetDIODirectionIn(DIONum)....14
SetDIODirectionOut(DIONum)....15
ReadDIO(DIONum)....15

ReadAD(ADNum)....15
ServoEnable(SvNum)....15
ServoDisable(SvNum)....15
ServoDisabledStateHigh(SvNum)....15
ServoDisabledStateLow(SvNum)....15
QuickMoveServoScaled(SvNum, SvPositionScaled)....15
QuickMoveServoPercent(SvNum, SvPositionPercent)....15
MoveServoScaled(SvNum, SvPositionScaled, SvSpeedPercent)....16
MoveServoPercent(SvNum, SvPositionPercent, SvSpeedPercent)....16
TimedMoveServoScaled(SvNum, SvPositionScaled, SvTime)....16
TimedMoveServoScaled(SvNum, SvPositionScaled, SvTime)....16
QuickLoadPreset(SceneNum)....17
CrossfadePreset(SceneNum,xFadeTimeTenths)....17
delaySec(timeSec)....17
delayMilliSec(timeMilliSec)....17
Srand(seed)....17
Rand( )....17
RandRange(rangeStart,rangeLast)....17
kbhit( )....17
getch( )....17
putch(AsciiValue)....17
CmdArg( )....18
TimeMSec( )....18
TimeSec( )....18
TimeMin( )....18
TimeClear( )....18
SetLedMode(ModeNumber)....18
3. ServoCenter Sequencer Control Commands....19
3.1 Protocol Command Summary....19
3.2 Protocol Command Details....19
4. Using the “ServoCenter Control Panel” Sequence Editor....20
4.1 Sequencer Programming Tab....21
4.2 Basic Use Tutorial....22
4.3 More Advanced Examples....23
5. SC-BASIC Programming Examples 23
5.1 Hello World 23
5.2 Hello World Sub....23
5.3 Servo Exercise....23
5.4 Servo Exercise With Sub 24
5.5 Digital I/O Exercise....24
5.6 Show all ADC Values....24
5.7 Move all Servos to a Random Position....24
5.8 Demo of global variable retaining value....25
5.9 Demo of CmdArg 25
6. Appendix....26
6.1 Hexadecimal/Decimal/Binary Nibble Conversion Chart....26
6.2 Hexadecimal / Decimal ASCII Chart....26
6.3 Token Specification....27
6.4 Error Codes....28

ServoCenter 4.1 Sequencer / SC-BASIC Language Interpreter

This document is intended to explain the details of the SC-BASIC language script sequencer that is used in the ServoCenter 4.1 servo controller boards.



1. Sequencer Overview

The ServoCenter 4.1 controller provides a built-in Sequencer / BASIC Interpreter called SC-BASIC that allows the ServoCenter board to be programmed to perform various tasks via the use of a simple tokenized BASIC-like scripting language. This feature can be used to implement customer specific logic, I/O, and servo control tasks without the need for a PC or other external command device.

The use of a fully functional scripting language allows the sequencer to perform complex logical and control tasks that can run independently on the SC4.1 board itself. The sequencer programs are stored on the SC4.1 board in a non-volatile EEPROM memory that retains the program even when the unit it reset or powered off. The loaded sequencer program can be started remotely via the “Start Sequencer” command and supports the passing of a parameter byte when a sequence is started. The sequencer also implements a “Sequencer Startup” feature that allows a loaded script to be executed when the SC4.1 board is reset or powered up thus allowing the SC4.1 board to be used as a stand-alone controller.

2. Sequencer Features

The SC-BASIC language was designed for the ServoCenter 4.1 Servo Controller as a simple, easy to learn language that is specifically suited to the architecture and problem domain of the controller board. SC-BASIC supports a syntax that will be familiar to users of other flavors of BASIC (such as QBASIC and Visual Basic) as well as some instructions and built-in functions that are specifically suited to the SC4.1 controller board.

SC-BASIC supports the following features:

- Compact tokenized program format.
- 4096 bytes of non-volatile program space.
- Familiar structured BASIC-like language syntax.
- Dynamic variable allocation.
- Support for up to 64 simultaneously allocated global and local integer variables.
- Support for up to 16 user-defined subroutines / functions.
- Scripts can be passed a parameter byte when started via the “Start Sequencer” command.
- Scripts can be started upon board reset / power up via the “Sequencer Startup” feature.
- Sequencer status and sequencer error reporting.
- Built-in servo control and digital I/O functions.
- Additional built-in functions for commonly needed programming tasks.
- Sequencer communication support allows the sequencer to print/receive messages.
- Sequencer Start / Stop / Reset control functions.

2. Language Reference

2.1 Comments

Comments provide a way of annotating and documenting source code. Also, since comments are ignored during execution, they provide a convenient means of temporarily commenting out code for debugging / testing purposes.

Comments start with the single quote character or pound sign character and continue to the end of the line. For example:

```
' This is a comment

# This is another comment

dim sv1 ' variable for servo position 1
```

2.2 Variables

2.2.1 Variable Basics

A variable is a way of assigning a name to a information stored in memory. SC-BASIC only supports integer variables. As such, each variable can hold an integer value in the range -32,768 to 32,767.

Variables are dynamically allocated, meaning that they are constantly created and destroyed as the program runs. Variables will not retain their value when the board is power-cycled or reset, or when the “Reset Sequencer” command is received.

SC-BASIC supports up to 64 simultaneously allocated variables.

2.2.2 Variable Scope and Lifetime

A variable's “scope” describes where a variable is accessible or visible to the program. A Variable's “lifetime” describes how long a variable exists before it is destroyed.

SC-BASIC supports two variable classifications that each have distinct lifetime / scope rules: global variables & procedure variables.

Global Variables

Global Variables are variables declared within the main program body – this means that these variables are not declared inside a Sub or Function. A global variable retains information until the board is power-cycles or reset, or until a “Reset Sequencer” command is received. A global variable is visible and usable anywhere within the program.

Procedure Variables

Procedure Variables are variables that are declared within a Sub or Function. Procedure variables only retain information and are only visible within the Sub or Function in which they are declared. This means that a procedure variables are created when they are declared within a sub or function and are automatically destroyed when that sub or function completes.

2.2.3 Variable Names

Variable names must adhere to the following naming rules:

- Variable names are not case sensitive.
- Must start with a letter
- Must be composed entirely of only letters, numbers, and the underscore character.
- Must not be a language keyword or built-in function name.

2.2.4 Declaring Variables

There are two ways to create variables for use by your script program. Variables can be declared implicitly or explicitly.

Implicit Declaration

Implicit variable declaration lets the program create the variable for you automatically. This is done by simply assigning a value to a legal variable name. For example:

```
count = 10
```

implicitly creates a variable named “count” and assigns the value 10 to it. Note that the left-side of the assignment operator (=) requires a variable name and the left side requires a valid expression.

Additionally, a variable will be implicitly created if it is used and hasn't previously been declared. For example:

```
count = 10 + y
```

will implicitly create two variables “count” and “y”. Variables that are created implicitly this way are initialized with a value of 0.

Explicit Declaration

Explicit variable declaration uses the Dim keyword to explicitly state that you want a variable created. For example:

```
Dim count
Dim y As Integer
```

declares two variable: “count” and y”. Note that the “As Integer” is optional and that variables that are declared without this are still created as integer variables.

Dim statements can be used at any time within the program, but are generally placed at the top of the program / sub / function in which the variable is declared.

2.2.5 Using Variables

Variable Assignment

Variables are assigned values using the assignment operator (=). For example:

```
i=10
count = 0
position = 1000
```

Variable Values

When the name of a variable is used in an expression, it is evaluated as the value that that variable is holding. For example:

```
i=10
count = 0
print i
count = count+1
position = count*10
```

2.3 Arithmetic Operations and Expressions

2.3.1 Arithmetic Operations

SC-BASIC supports the following arithmetic operations:

Operator	*Name*	*Use*
( )	Parenthesis	Group other operations
+	Addition	Add values
-	Subtraction or Unary Negation	Subtract values or negate a value
*	Multiplication	Multiply values
/	Division	Divide values
^	Exponentiation	Raise a value to a power
Mod	Modulus	Determine the remainder of a division.

Note that since SC-BASIC only supports integer data types, division will result in only the integer division result. Additionally, floating-point values and constants are not legal.

2.3.2 Arithmetic Order of Operations

SC-BASIC implements standard algebraic order of operations as follows:

Order	*Operators*	*Name*
1	( )	Parenthesis
2	-	Unary Negation (Negative)
3	^	Exponentiation
4	*, /, Mod	Multiplication, Division, Modulus from Left to Right
5	+, -	Addition and Subtraction form Left to Right

2.3.3 Arithmetic Operation Examples

Operation	Result
5+10	15
15-14	1
14-15	-1
3*6	18
3*-6	-18
10/3	3
3/10	0
2^4	16
10 Mod 3	1

2.3.4 Arithmetic Expressions

Many SC-BASIC commands use expressions. A valid arithmetic expression is formed from any combination of the following:

1. Integer numeric literals.
2. Correctly used arithmetic operators
3. Variable names.
4. Correctly formed function calls.
5. Correctly used logical operators

Logical operators and function calls are explained in further detail later in this document.

2.4 Relational and Logical Operations

Relational operators provide a way to get SC-BASIC programs to perform standard numeric comparisons.

Logical operators provide a way to get SC-BASIC programs to perform the standard boolean logical operations And, Or, Not.

These are particularly useful in decision statements and looping statements discussed later in this document.

SC-BASIC uses a value of 0 to represent false and a non-zero value to represent true.

2.4.1 Relational Operators

SC-BASIC supports the following relational operations:

Operator	*Name*	*Use*
=	Equals	True if two expressions are equal in value
>	Greater-than	True if left-side expression is greater than right-side
<	Less-than	True if left-side expression is less than right-side
>=	Greater-or-equal-to	True if left-side expression is greater or equal to right-side
<=	Less-or-equal-to	True if left-side expression is less or equal to right-side
<> , !=	Not-equal-to	True if two expressions are not equal in value

2.4.2 Logical Operators

SC-BASIC supports the following logical operations:

Operator	*Name*	*Use*
Or	Logical Or	True if either left-side or right-side expression is true
And	Logical And	True if both left-side and right-side expression are true
Not	Logical Unary Not	True is following expression is false

2.4.3 Relational / Logical Order of Operations

SC-BASIC evaluates expressions using the following order of operations: arithmetic expressions followed by relational and logical expressions. Standard relational and logical order of operations as follows:

Order	*Operators*	*Name*
1	=,<,>,<=,>=,<>	Relational Operators from left to right
2	Not	Unary Logical Negation
3	And	Logical And Operators from Left to Right
4	Or	Logical Or Operators from Left to Right

2.4.4 Relational / Logical Operation Examples

Operation	Result
1=1	1 (true)
5>10	0 (false)
10<15	1 (true)
1<2 And 2>1	1 (true)
1=0 Or 0>1	0 (false)
Not (1=1)	0 (false)

2.5 Language Statements

Language statements are detailed in the following sections. Note that language statements are not case sensitive.

2.5.1 Print Statement

Print Statement Details

The Print statement allows a SC-BASIC program to print information to the communication serial or USB port.

Caution must be used when using this feature since the program printing directly to the port could affect or corrupt other communication message data that is being requested from the SC4.1 boards.

The print statement accepts a sequence of arguments that are each separated by a comma or a semicolon. Argument items can be any of the following:

1. Literal strings. (“hello world”, “program done”, etc.)
2. Variable names
3. Numeric literals
4. Logical expressions
5. Arithmetic expressions

When a comma is used as an argument separator, a TAB(ASCII HT) character is inserted between the output of the arguments.

When a semicolon is used as an argument separator, items are printed without any space between subsequent arguments.

The print statement normally prints carriage return and line-feed characters at the end of each print lines automatically. This behavior is suppressed if the print statement line ends with a semicolon or a comma.

When print is used on a line by itself with no arguments, a blank line (CRLF) is printed.

Print Statement Examples

```
Print "Hello World"
Print position
Print "count is:",count
Print "Count by tens:",10,20,30
Print "six digits:",10;20;30
Print "Average:",(10+20+30)/3
Print "First line",
Print "same line"
Print "All";
Print "Together"
```

2.5.2 If Statements

If statements allow the SC_BASIC language to make decisions based upon an evaluated expression.

SC-BASIC supports the same standard if statement forms as other commonly encountered BASIC interpreters:

Single Line If Statement Forms

Single line If statements can have one of two forms:

Form 1: `If [expression] Then [command line to execute]`

Form 2: `If [expression] Then [command line to execute] Else [command line to execute]`

Block If Statement Forms

Block If statements can have one of three forms.

Form 1:
```
If [expression] Then
    [multiple lines of code to execute if true]
End If
```

Form 2:
```
If [expression] Then
    [multiple lines of code to execute if true]
Else
    [multiple lines of code to execute if false]
End If
```

Form 3:
```
If [expression1] Then
    [multiple lines of code to execute if expression1 true]
ElseIf [expression2] Then
    [multiple lines of code to execute if expression2 true]
ElseIf [expression3] Then
    [multiple lines of code to execute if expression3 true]
Else
    [multiple lines of code to execute if all expressions are false]
End If
```

2.5.3 For / Next Loops

For Next Loop statements are a counting loop construct that allows the SC_BASIC language to repeat a section of code a specific number of times by counting through a sequence of values. The counting sequence uses a specified variable to keep track of the count and counts from the specified start value to the specified end value, counting by the optional step value provided.

For / Next Statement Forms

Form 1:
```
For [variable] = [start expression] To [end expression]
        [multiple lines of code to repeat]
Next [variable]
```

Form 2:
```
For [variable] = [start expression] To [end expression] Step [step count expression]
        [multiple lines of code to repeat]
Next [variable]
```

Exit For Statement

The “Exit For” statement, when encountered within an executing For Next loop, immediately causes the termination of the innermost For Next loop. Program execution resumes following the exited loop's Next statement.

2.5.4 Do While Loops

For Do While statements are a conditional loop construct that allows the SC_BASIC language to repeat a section of code while an expression is evaluated as true. If the conditional expression is evaluated as false, execution resumes following the corresponding Loop statement.

Do While / Loop Forms

SC-BASIC provides slightly different While keyword forms to preserve familiarity with other versions of BASIC.

Form 1:
```
Do While [expression]
        [multiple lines of code to execute while expression is true]
Loop
```

Form 2:
```
While [expression]
        [multiple lines of code to execute while expression is true]
Wend
```

Form 3:
```
While [expression]
        [multiple lines of code to execute while expression is true]
End While
```

Exit While Statement

The “Exit While” statement, when encountered within an executing While loop, immediately causes the termination of the innermost While Loop. Program execution resumes following the exited loop's Loop/Wend/End While statement.

Infinite Loops

It is possible to make an infinite loop in SC-BASIC by ensuring that the condition always evaluates as true. This is most easily accomplished by simply inserting a non-zero number as the conditional expression in the do while statement. For example:

```
Do While 1 ' condition is always true
    Print "to infinity..."
Loop
```

or

```
Do While 1=1 ' condition is always true
    Print "to infinity"
Loop
```

Note that infinite loops can still be exited by using the "Exit While" statement.

2.5.5 Procedures: Subs and Functions

Procedure Basics

A procedure is a defined section of code that can be called upon to perform a particular task. Procedures are defined by associating a procedure name with code that is outside of the main execution area. A define procedure can be called to invoke the execution of its defined code. Procedures can have data passed to them when invoked via the use of parameters (sometimes called arguments). Function procedures can return data upon completion.

SC-BASIC provides support for two types of procedures:

1. Sub Procedures – A procedure that doesn't return a value.
2. Function Procedures – A procedure that returns a value.

Thus, the only real difference between Sub procedures and Function procedures is the ability of functions to return a value.

In SC-BASIC functions and procedures can be defined either before or after the main code block, thus freeing the programmer to organize code in whichever way is most convenient.

Naming rules for procedures follow the same conventions as variable names.

Defining a Sub

A Sub procedure is defined using the Sub / End Sub statements. For example:

```
Sub Hello()
  Print "Hello World"
End Sub
```

The above example defines a Sub procedure named "Hello" that prints out the message "Hello World".

Calling a Sub

A Sub procedure can be invoked in two ways: Using the Call statement, or using the name of the procedure.

Method 1: `Call Hello()`

Method 2: `Hello()`

Passing Information to a Sub

Often it is desirable to send information to a procedure to tell it how or what to do. This information is, as a group, called the parameter list or argument list of the procedure.

The procedure definition identifies and references these data items by using variable names.

For example the Sub defined as:

```
Sub HelloMany( count )
    For i = 1 to count
        Print "Hello World"
    Next i
End Sub
```

Allows a single parameter “count” is defined that allows a number to passed to the procedure.

Multiple parameters can be passed to a function, but since each parameter uses a local variable, care must be taken so as not to exceed the capacity of the local variable table. When passing multiple parameters, a comma separated list of parameters is included in the procedure's definition.

Parameter lists support the use of type specification via the typical “As Integer” specifier, but integer types are exclusively supported.

Parameters in SC-BASIC are exclusively passed by value and there is no support for parameters passed by reference. Global variables should be considered if a procedure must return multiple values.

When a procedure is called, the number of parameters passed to the function must match the number of parameters listed in the procedure definition.

Exit Sub Statement

The “Exit Sub” statement, when encountered within an executing Sub procedure, immediately causes the termination of the procedure call. Program execution resumes following the statement that invoked the call.

Defining a Function

A Function procedure is defined using the Function / End Function statements. For example:

```
Function Midpoint( num1, num2 )
    Midpoint = (num1+num2)/2
End Function
```

The above example defines a Function procedure named “Midpoint” that accepts two parameter values and returns the midpoint between those two numbers.

Specifying Function Return Values

Functions in SC-BASIC can specify their return value by assigning a value to the name of the function within the function body. When the function terminates by encountering the End Function or Exit Function statements, the value assigned to the name of the function is returned. If no return value is explicitly assigned, a value of 0 is returned.

Calling a Function

A Function procedure can be invoked in several ways, but generally, the function call is embedded within an arithmetic expression. For example:

```
xmid = Midpoint(x1,x2)
```

or

```
Print "Midpoint is:",Midpoint(x1,x2)
```

Thus, a function can be used in any context a variable can be used.

Functions can also be called identically to the methods of calling Sub procedures. For example:

```
DelaySec(3)
```

or

```
Call DelaySec(3)
```

Exit Function Statement

The “Exit Function” statement, when encountered within an executing function procedure, immediately causes the termination of the function. The currently set function return value is returned to the point where the function call was made. If no return value had been set, a 0 is returned.

2.5.6 The End Statement

The “End” statement causes the executing program to immediately halt execution.

2.6 Built-in Functions

SC-BASIC includes many built-in functions that complement the capabilities of the ServoCenter4.1 family of controller boards. Built-in functions are not case-sensitive.

Pass( )

The Pass( ) function simply returns a value of 0. This can be useful when a place-holder function is needed during testing or debugging.

SetDIOHigh(DIONum)

The SetDIOHigh(DIONum) function sets the digital I/O pin specified by the DIONum parameter (0~15) to a “high” (logic 1) state. If the port is configured as an input, the pin's internal pull-up resistor is activated. The SetDIOHigh function always returns 0.

SetDIOLow(DIONum)

The SetDIOLow(DIONum) function sets the digital I/O pin specified by the DIONum parameter (0~15) to a “low” (logic 0) state. If the port is configured as an input, the pin's internal pull-up resistor is deactivated. The SetDIOLow function always returns 0.

SetDIODirectionIn(DIONum)

The SetDIODirectionIn(DIONum) function sets the digital I/O pin specified by the DIONum parameter (0~15) to act as an input. The SetDIODirectionIn function always returns 0.

SetDIODirectionOut(DIONum)

The SetDIODirectionOut(DIONum) function sets the digital I/O pin specified by the DIONum parameter (0~15) to act as an output. The SetDIODirectionOut function always returns 0.

ReadDIO(DIONum)

The ReadDIO(DIONum) function reads and returns the state of the digital I/O pin specified by the DIONum parameter (0~15). The ReadDIO function returns the logic state (0,1) of the specified DIO pin.

ReadAD(ADNum)

The ReadAD(ADNum) function reads and returns the 10-bit conversion value from the ADC pin specified by the ADNum parameter (0~7). The ReadAD function returns 10-bit conversion value (0~1023) of the specified ADC pin.

ServoEnable(SvNum)

The ServoEnable(SvNum) function enables the control signal on the servo channel specified by the SvNum parameter (0~15). The ServoEnable function returns 0.

ServoDisable(SvNum)

The ServoDisable(SvNum) function disables the control signal on the servo channel specified by the SvNum parameter (0~15). A disabled servo channel's logical state state is determined by the servo channel's “Disabled State”. The ServoDisable function always returns 0.

ServoDisabledStateHigh(SvNum)

The ServoDisabledStateHigh(SvNum) function sets the disabled logical state of the servo channel specified by the SvNum parameter (0~15) to high (logic 1). The ServoDisabledStateHigh function always returns 0.

ServoDisabledStateLow(SvNum)

The ServoDisabledStateLow(SvNum) function sets the disabled logical state of the servo channel specified by the SvNum parameter (0~15) to low (logic 0). The ServoDisabledStateLow function always returns 0.

QuickMoveServoScaled(SvNum, SvPositionScaled)

The QuickMoveServoScaled(SvNum, SvPositionScaled) immediately sets the position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionScaled (0~16383). Position 0 is the servo channel's defined min position, position 16383 is the servo channel's defined max position, values between 0 and 16383 are linearly interpolated between the defined min and max positions. The QuickMoveServoScaled function always returns 0.

QuickMoveServoPercent(SvNum, SvPositionPercent)

The QuickMoveServoPercent(SvNum, SvPositionPercent) immediately sets the position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionPercent (0~10000). Positions are in 100ths of a percent of the full-scale min-to-max range. Thus, position 0 is the servo channel's defined min position, position 10000 is the servo channel's defined max position, values between 0 and 10000 are linearly interpolated between the defined min and max positions. The QuickMoveServoPercent function always returns 0.

MoveServoScaled(SvNum, SvPositionScaled, SvSpeedPercent)

The MoveServoScaled(SvNum, SvPositionScaled, SvSpeedPercent) moves the servo position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionScaled (0~16383) according to the speed determined by SvSpeedPercent (0-10000). Position 0 is the servo channel's defined min position, position 16383 is the servo channel's defined max position, values between 0 and 16383 are linearly interpolated between the defined min and max positions. SvSpeedPercent is in 100ths of a percent of the speed defined for the specified servo channel. Thus a speed of 10000 will move the servo instantly to the defined positions, 5000 will move at half the full speed, 1000 will move at 1/10th full speed, etc. The MoveServoScaled function always returns 0.

MoveServoPercent(SvNum, SvPositionPercent, SvSpeedPercent)

The MoveServoPercent(SvNum, SvPositionPercent, SvSpeedPercent) moves the servo position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionPercent (0~10000) according to the speed determined by SvSpeedPercent (0-10000). Positions are in 100ths of a percent of the full-scale min-to-max range. Thus, position 0 is the servo channel's defined min position, position 10000 is the servo channel's defined max position, values between 0 and 10000 are linearly interpolated between the defined min and max positions. SvSpeedPercent is in 100ths of a percent of the speed defined for the specified servo channel. Thus a speed of 10000 will move the servo instantly to the defined positions, 5000 will move at half the full speed, 1000 will move at 1/10th full speed, etc. The MoveServoPercent function always returns 0.

TimedMoveServoScaled(SvNum, SvPositionScaled, SvTime)

The TimedMoveServoScaled(SvNum, SvPositionScaled, SvTime) moves the servo position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionScaled (0~16383) and takes the amount of time specified by SvTime(0~16383). Position 0 is the servo channel's defined min position, position 16383 is the servo channel's defined max position, values between 0 and 16383 are linearly interpolated between the defined min and max positions. SvTime is in 100ths of a second. Thus a speed of 1000 will take 10 seconds, a speed of 300 will take 3 seconds, 100 will take 1 second, etc. The TimedMoveServoScaled function always returns 0.

TimedMoveServoScaled(SvNum, SvPositionScaled, SvTime)

The TimedMoveServoScaled(SvNum, SvPositionPercent, SvTime) moves the servo position of the servo channel specified by SvNum (0~15) to the scaled position specified by SvPositionPercent (0~10000) and takes the amount of time specified by SvTime(0~16383). Positions are in 100ths of a percent of the full-scale min-to-max range. Thus, position 0 is the servo channel's defined min position, position 10000 is the servo channel's defined max position, values between 0 and 10000 are linearly interpolated between the defined min and max positions. SvTime is in 100ths of a second. Thus a speed of 1000 will take 10 seconds, a speed of 300 will take 3 seconds, 100 will take 1 second, etc. The TimedMoveServoScaled function always returns 0.

QuickLoadPreset(SceneNum)

The QuickLoadPreset(SceneNum) immediately loads the preset “scene” indicated by the SceneNum parameter (0~63). Scenes provide a method of storing and loading preset servo and digital IO configurations. The QuickLoadPreset function always returns 0.

CrossfadePreset(SceneNum,xFadeTimeTenths)

The CrossfadePreset(SceneNum, xFadeTimeTenths) loads the preset “scene” indicated by the SceneNum parameter (0~63), but smoothly crossfades the scene's servo positions from current positions, taking the amount of time indicated by the xFadeTimeTenths parameter. Scenes provide a method of storing and loading preset servo and digital IO configurations. The CrossfadePreset function always returns 0.

delaySec(timeSec)

The delaySec(timeSec) function pauses execution of the running program for the amount of seconds indicated by timeSec (0~32767). The delaySec function always returns 0.

delayMilliSec(timeMilliSec)

The delayMilliSec(timeMilliSec) function pauses execution of the running program for the amount of milliseconds indicated by timeMilliSec (0~32767). The delayMilliSec function always returns 0.

Srand(seed)

The Srand(seed) function seeds the random number generator with the value indicated by seed (0~32767). The Srand function always returns 0.

Rand( )

The Rand( ) function returns a pseudo-random integer in the range [0~32767].

RandRange(rangeStart,rangeLast)

The RandRange(rangeStart,rangeLast) function returns a pseudo-random integer in the range specified by [rangeStart ~ rangeLast].

kbhit( )

The kbhit( ) function returns the number of characters waiting in the sequencer input buffer. 0 is returned if the buffer is empty. Note that characters are placed in the buffer by ServoCenter command 213(0xd5) “Write Character to Sequencer”. The sequencer communication buffer is 16 characters in size.

getch( )

The getch( ) function returns the next character waiting in the sequencer input buffer. If the buffer is empty, the function blocks and awaits a character. Note that characters are placed in the buffer by ServoCenter command 213(0xd5) “Write Character to Sequencer”. The sequencer communication buffer is 16 characters in size.

putch(AsciiValue)

The putch(AsciiValue) function writes the character specified by AsciiValue to the output communication stream. Note that this write is unbuffered so care must be taken so as not to interrupt other communication messages or requests being communicated. The putch function always returns 0.

CmdArg( )

The CmdArg( ) function returns the command argument byte value that was set when the sequencer program was started with the ServoCenter command 210(0xd2) “Start Sequencer”. This gives a way of controlling how or what the program does upon execution. A program that is started automatically on power-up or reset due to the sequencer startup mode will have a CmdArg of 0.

TimeMSec( )

The TimeMSec( ) function returns the millisecond portion of the time clock. The time clock measures the amount of time since the board was last power-cycled, reset, or had the time explicitly cleared. Unlike the delay functions, the time functions do not pause execution, but simply return time clock readings. The time clock runs even when the sequencer is stopped or idle. The time functions can be useful for measuring time intervals or triggering events based upon time passage.

TimeSec( )

The TimeSec( ) function returns the second portion of the time clock. The time clock measures the amount of time since the board was last power-cycled, reset, or had the time explicitly cleared. Unlike the delay functions, the time functions do not pause execution, but simply return time clock readings. The time clock runs even when the sequencer is stopped or idle. The time functions can be useful for measuring time intervals or triggering events based upon time passage.

TimeMin( )

The TimeMin( ) function returns the minutes portion of the time clock. The time clock measures the amount of time since the board was last power-cycled, reset, or had the time explicitly cleared. Unlike the delay functions, the time functions do not pause execution, but simply return time clock readings. The time clock runs even when the sequencer is stopped or idle. The time functions can be useful for measuring time intervals or triggering events based upon time passage.

TimeClear( )

The TimeClear( ) function sets the millisecond, second, and minute values of the time clock to 0. Unlike the delay functions, the time functions do not pause execution, but simply return time clock readings. The time clock runs even when the sequencer is stopped or idle. The time functions can be useful for measuring time intervals or triggering events based upon time passage.

SetLedMode(ModeNumber)

The SetLedMode(ModeNumber) function sets the LED display mode for the on-board indicators LED1 and LED2 to the value indicated by ModeNumber(0~7). This can be useful for troubleshooting purposes or specifying a desired LED output. The default settings is mode 4. Mode values are as follows:

Mode 0: LED2 =off , LED1=off
Mode 1: LED2 =off , LED1=on
Mode 2: LED2 =on , LED1=off
Mode 3: LED2 =on , LED1=on
Mode 4: LED2=statAction , LED1=statRx
Mode 5: LED2=statServoAction , LED1=statRx
Mode 6: LED2=statServoAction , LED1=statAction
Mode 7: reserved

3. ServoCenter Sequencer Control Commands

The ServoCenter controller provides several commands that are related to the SC-BASIC sequencer script programming.

3.1 Protocol Command Summary

Description	Command	Data Len	Data Descriptions
Read Sequencer EEPROM Page	208 (0xd0)	1	EepromPageNum(0~127)
Write Sequencer EEPROM Page	209 (0xd1)	33	EepromPageNum(0~127), 32 x [EepromDataByte]
Start Sequencer	210 (0xd2)	1	SequencerArg(0~255)
Stop Sequencer	211(0xd3)	0	
Reset Sequencer	212 (0xd4)	0	
Write Character to Sequencer	213 (0xd5)	1	DataByte(0~255)
Set Sequencer Startup Mode	214 (0xd6)	1	SequenceStartupMode(0,1)
Get Sequencer Startup Mode	215 (0xd7)	0	
Get Sequencer Status	216 (0xd8)	0	
Get Sequencer Last Error	217 (0xd9)	0	

3.2 Protocol Command Details

Function:	Read Sequencer EEPROM Page
Command Value:	208 (0xd0)
Data Bytes:	1
Data Format:	EepromPageNum(0~127)
Description:	The Read Sequencer EEPROM Page allows the contents of the EEPROM sequencer memory to be read one page at a time. Each page consists of 32 bytes of non-volatile EEPROM data. The data within the EEPROM is a tokenized P-Code program for the SC4.1 BASIC Interpreter.

Function:	Write Sequencer EEPROM Page
Command Value:	209 (0xd1)
Data Bytes:	33
Data Format:	EepromPageNum(0~127), 32 x [EepromDataByte]
Description:	The Write Sequencer EEPROM Page allows the contents of the EEPROM sequencer memory to be written one page at a time. Each page consists of 32 bytes of non-volatile EEPROM data. The data within the EEPROM is a tokenized P-Code program for the SC4.1 BASIC Interpreter.

Function:	Start Sequencer
Command Value:	210 (0xd2)
Data Bytes:	1
Data Format:	SequencerArg(0~255)
Description:	The Start Sequencer command causes the execution of the program stored within the EEPROM sequencer memory. When the script is started a single byte parameter argument (SequencerArg) can be passed to the script. This allows a script to be started in different ways by the Start Sequencer command. When a program is started, it always begins at the beginning of the script, and runs to completion, until stopped by the Stop Sequencer or Reset Sequencer commands, or until a sequencer error is encountered. Although execution always begins at the beginning of the script, variables declared within the global scope are not cleared when a program is started. This allows a program to “remember” information from one run to the next if desired. While the sequencer is running a program the ServoCenter board can still respond to other control messages. Care must be taken to avoid having these incoming control messages interfere with the programmed operation of the sequencer program. For example the sequencer could be programmed to move Servo S0 to position 0 and an incoming command could instruct Servo S0 to move to position 100. In cases such as this, the most recently issued command will take precedence. Additionally, be aware that the sequencer has instructions that can transmit, or print, data from the SC4.1 board to the controller. This could cause mis-interpretation of responses when issuing commands that get data from the controller so care must be taken when reading from a SC4.1 board while the sequencer is running a program that transmits characters or uses the print statement.

Function:	Stop Sequencer
Command Value:	211 (0xd3)
Data Bytes:	0
Data Format:	
Description:	The Stop Sequencer command causes any running sequencer program to immediately halt execution. Note that when a program is stopped, variables declared in the global scope are not cleared. This allows a program to “remember” information from one run to the next if desired.

Function:	Reset Sequencer
Command Value:	212 (0xd4)
Data Bytes:	0
Data Format:	
Description:	The Reset Sequencer command causes any running sequencer program to immediately halt execution and the contents of all globally declared variables are cleared.

Function:	Write Character to Sequencer
Command Value:	213 (0xd5)
Data Bytes:	1
Data Format:	DataByte(0~255)
Description:	The Write Character to Sequencer command causes a single byte of data specified by DataByte to be passed to the sequencer's character input buffer. This provides a means of communicating with and controlling a running sequencer program. The sequencer program can read and respond to these messages by using the built-in getch( ) and kbhit( ) functions.

Function:	Set Sequencer Startup Mode
Command Value:	214 (0xd6)
Data Bytes:	1
Data Format:	SequenceStartupMode(0,1)
Description:	The Set Sequencer Startup Mode command allows the setting of the sequencer startup mode to one of the following: 0 – The stored sequencer program can only be started by the reception of the Start Sequencer command. 1 – The stored sequencer program is automatically started when the board is reset / powered up or when the Start Sequencer command is received. This effectively allows the SC4.1 to be used in stand-alone applications.

Function:	Get Sequencer Startup Mode
Command Value:	215 (0xd7)
Data Bytes:	0
Data Format:	
Description:	The Get Sequencer Startup Mode command allows the reading of the sequencer startup mode. Valid startup modes are: 0 – The stored sequencer program can only be started by the reception of the Start Sequencer command. 1 – The stored sequencer program is automatically started when the board is reset / powered up or when the Start Sequencer command is received. This effectively allows the SC4.1 to be used in stand-alone applications.

Function:	Get Sequencer Status
Command Value:	216 (0xd8)
Data Bytes:	0
Data Format:	
Description:	The Get Sequencer Status command allows the reading of the sequencer status byte. The sequencer status byte values are: 0 – Sequencer Idle / Stopped 1 – Sequencer Running 2 – Sequencer Error / Stopped

Function:	Get Sequencer Last Error
Command Value:	217 (0xd9)
Data Bytes:	0
Data Format:	
Description:	The Get Sequencer Last Error command allows the reading of the sequencer error code byte of the last error that the sequencer encountered. The sequencer error byte values are described in detail in appendix 6.4 “Error Codes”.

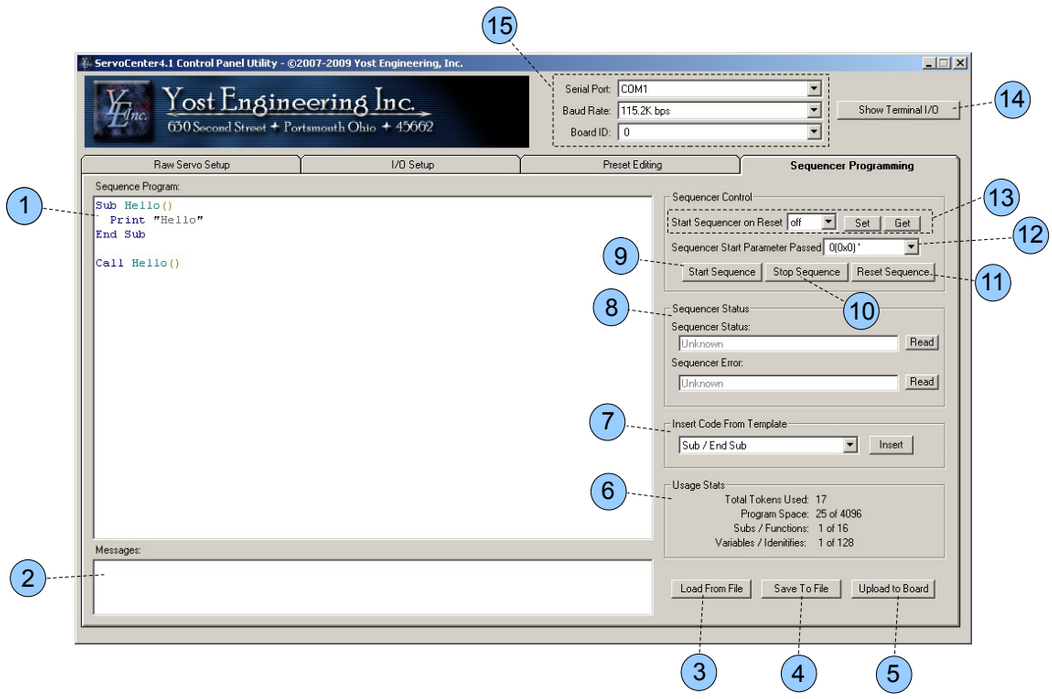
4. Using the “ServoCenter Control Panel” Sequence Editor

The “ServoCenter Control Panel” application allows for easy editing and configuration of many aspects of the ServoCenter 4.1 servo controller. This includes an SC-BASIC program editor that allows the writing / compiling / uploading / testing of SC-BASIC programs.

The sections below describe the use of the SC-BASIC-related aspects of the Control Panel application.

4.1 Sequencer Programming Tab

To access the SC-BASIC-related features of the "Control Panel" application, click on the Sequencer Programming tab.



1. **Code Editor window** – This window allows you to type and edit SC-BASIC programs.
2. **Message window** – This window shows errors and warnings generated by the SC4.1 tokenizer. Only lexical errors are reported, logic, semantic, and syntax errors are reported when the program is executed.
3. **Load From File button** – This button allows programs to be loaded into the editor windows from file.
4. **Save to File button** – This button allows programs in the editor window to be saved to file.
5. **Upload to Board button** – This button allows the tokenized code to be uploaded to the SC4.1 controller's non-volatile EEPROM program storage.
6. **Usage Statistics information pane** – This area is used to display the program memory, variable, and procedure usage information about the program code that is in the editing window.
7. **Code Assistant pane** – This area allows the selection of commonly used code constructs and built-in functions. Code is inserted at the location of the cursor when the Insert button is clicked.
8. **Sequencer Status pane** – This area allows the reading and display of the sequencer's running status and last error message. These are individually read when the respective Read button is pressed.
9. **Start Sequence button** – This button sends the start sequencer protocol command to the attached controller board and passes the start parameter byte selected in the "Start Parameter" pull-down (see 12). Note that a program must be uploaded (see 5). before it can be started.
10. **Stop Sequencer button** – This button sends the stop sequencer protocol command to the controller board. If a program is running, it is immediately halted. Variables are not cleared.

11. **Reset Sequencer button** - This button sends the reset sequencer protocol command to the controller board. If a program is running, it is immediately halted and all variables are cleared.
12. **Start Parameter selection** – This selection box allows the selection of the parameter byte that is sent to the controller when the Start Sequence button is clicked.
13. **Sequencer start mode control** – These controls allow the setting of the sequencer's start mode. When "on" the sequencer will execute the loaded program whenever the controller board is reset or powered up. This can allow the controller board to be used without any external connections. When "off" the sequencer must be started by sending the "Start Sequencer" protocol command.
14. **Show Terminal I/O button** - This allows the display of the terminal snooper that allows the inspection of outgoing and incoming communications messages. This is especially useful for monitoring the communication output of SC-BASIC programs.
15. **Communication Setup** – These controls allow the configuration of port / baud / and board id settings for the board that is to be controlled.

4.2 Basic Use Tutorial

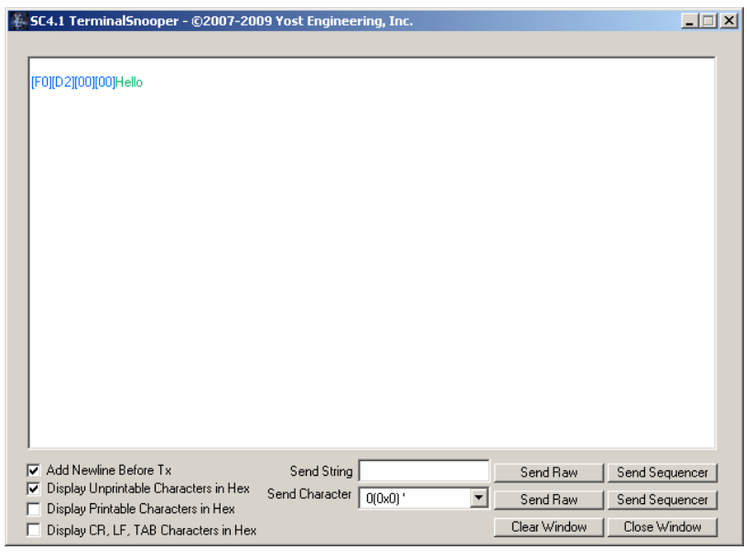
To write and test a program, follow these steps:

1. Type a program in the **Code Editor** window. Try the following program:

```
Sub Hello()
  Print "Hello"
End Sub

Call Hello()
```

2. If desired save the program using the "Save to File" button.
3. Upload the program to the board using the "Upload to Board" button. If the program uploaded correctly click Ok to dismiss the dialog. If the program didn't upload successfully, check the communications port / baud / board Id settings.
4. Bring up the terminal window by clicking the "Show Terminal I/O" button.
5. Start the program by clicking the "Start Sequence" button.
6. Observe the output in the Terminal I/O window. The window should look like this:



Note that the communication messages being sent are shown in blue and the messages being received are shown in green.

4.3 More Advanced Examples

Section 5 contains several additional examples that each illustrate different features of the SC-BASIC language. It is recommended that these examples be analyzed and understood before trying your own programs and sequences.

5. SC-BASIC Programming Examples

5.1 Hello World

```
' Simple Hello World Program

Print "Hello World"
```

5.2 Hello World Sub

```
'Hello World Program using a Sub Procedure.

Sub Hello()
      Print "Hello World"
End Sub

Call Hello()
```

5.3 Servo Exercise

```
' This program exercises all 16 servo channels by
'  moving them from min to max to min to center
'  with a 1 second delay between each movement.

' move all servos to Min positions
for SvNum = 0 to 15
    QuickMoveServoScaled(SvNum,0)
next SvNum

' wait for a second
Call DelaySec(1)

' move all servos to Max positions
for SvNum = 0 to 15
    QuickMoveServoScaled(SvNum,16383)
next SvNum

' wait for a second
Call DelaySec(1)

' move all servos to Min positions
for SvNum = 0 to 15
    QuickMoveServoScaled(SvNum,0)
next SvNum

' wait for a second
Call DelaySec(1)

' move all servos to center positions
for SvNum = 0 to 15
    QuickMoveServoScaled(SvNum,8191)
next SvNum
```

5.4 Servo Exercise With Sub

```
' This program exercises all 16 servo channels by
'  moving them from min to max to min to center
'  with a 1 second delay between each movement.

Sub MoveAllServos(Pos)
    'move all servos to position Pos
    for SvNum = 0 to 15
        QuickMoveServoScaled(SvNum,Pos)
    next SvNum
End Sub

'move all servos to Min positions
Call MoveAllServos(0)
'wait for a second
Call DelaySec(1)

'move all servos to Max positions
Call MoveAllServos(16383)
'wait for a second
Call DelaySec(1)

'move all servos to Min positions
Call MoveAllServos(0)
'wait for a second
Call DelaySec(1)

'move all servos to Center positions
Call MoveAllServos(8191)
```

5.5 Digital I/O Exercise

```
' This program exercises all 16 Digital I/O channels by
'  setting them all as outputs and cycling through each one
'  with a 10th of a second high pulse.


' Set all Digital I/O pins as output
For DioNum = 0 To 15
    SetDIODirectionOut(DioNum)
    Call SetDIOLow(DioNum)
Next DioNum

' Cycle through all 16 digital I/O pins setting each high for 1/10th
'  of a second then low.
For DioNum = 0 To 15
    Call SetDIOHigh(DioNum)
    Call DelayMilliSec(100)
    Call SetDIOLow(DioNum)
Next DioNum

print "done"
```

5.6 Show all ADC Values

```
' This program prints the ADC values for all 8 ADC channels.

For AdNum = 0 to 7
    Print "ADC Channel ";AdNum;" reads: ";ReadAD(AdNum)
next AdNum
```

5.7 Move all Servos to a Random Position

```
 ' This program moves all servos to a random position.

For SvNum = 0 To 15
  Call QuickMoveServoScaled(SvNum,RandRange(0,16383))
Next SvNum
```

5.8 Demo of global variable retaining value

```
'  The following program demonstrates the ability of SC-BASIC to retain
'    global variable values between executions.  Run the program multiple times
'    to see the bahavior.   To prevent this behavior, initialize variables before
'    use, or issue the "Reset Sequencer" command before issuing the
'    "Start Sequencer" command.

 print "The variable count is:",count

count=count+1
```

5.9 Demo of CmdArg

```
' This program illustrates the use of the CmdArg function to control
'    the behavior of a program when it is started.

choice = CmdArg()

If choice=0 Then
    Print "CmdArg was 0"
Else If choice=1 Then
    Print "CmdArg was 1"
Else If choice = 2 Then
    Print "CmdArg was 2"
Else If choice = 3 Then
    Print "CmdArg was 0"
Else
    Print "CmdArg was greater than 3"
End if
```

6. Appendix

6.1 Hexadecimal/Decimal/Binary Nibble Conversion Chart

Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

6.2 Hexadecimal / Decimal ASCII Chart

ASCII	HEX	Symbol	ASCII	HEX	Symbol	ASCII	HEX	Symbol	ASCII	HEX	Symbol
0	0	NUL	32	20	(space)	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

6.3 Token Specification

It is possible to program the ServoCenter4.1 SC-BASIC sequencer directly using byte-code representations of the program and directly loading them into the sequencer memory using the “Write Sequencer EEPROM Page” protocol command. The following section describes each of the low-level byte tokens as they are stored in memory.

Character	Byte Value	Token Name	Description	Lexical Rule	Note
'_'		T_ERROR	Lexical / scanning error	None	
' '	32	T_NONE	No operation	None	
'\0'	0	T_EOFI	End of File / End of Input	(\0' | End of File)	
'\n'	10	T_EOL	End of Line	('\r' | '\n')+	
'\r'	13	T_EOL	End of Line	('\r' | '\n')+	
'+'	43	T_ADD	Addition / unary plus	'+'	
'-'	45	T_SUB	Subtraction / unary negative	'-'	
'/'	47	T_DIV	Division	'/'	
'*'	42	T_MUL	Multiplication	'*'	
'%'	37	T_MOD	Modulus	'%' | 'Mod'	
'^'	94	T_EXP	Exponentiation	'^'	
'('	40	T_LPAREN	Left Parenthesis	'('	
')'	41	T_RPAREN	Right Parenthesis	')'	
'='	61	T_EQUAL	Equal Sign	'='	
','	44	T_COMMA	Comma	','	
';'	59	T_SEMI	Semicolon	';'	
':'	58	T_COLON	Colon	':'	
'<'	60	T_LT	Less than	'<'	
'>'	62	T_GT	Greater than	'>'	
'['	91	T_LE	Less than or Equal to	'<='	
']'	93	T_GE	Greater than or Equal to	'>='	
'X'	88	T_NE	Not Equal	('<>' | '!=')	
'&'	38	T_AND	Logical And Operator	'and'	
'|'	124	T_OR	Logical Or Operator	'or'	
'!'	33	T_NOT	Not Operator	'not'	
'P'	80	T_PRINT	Print command token	'print'	
'I'	73	T_IF	If command token	'if'	
'T'	84	T_THEN	Then command token	'then'	
'e'	101	T_ELSEIF	ElseIf command token	('elseif' | 'else if')	
'i'	105	T_ENDIF	EndIf command token	('endif' | 'end if')	
'F'	70	T_FOR	For command token	'for'	
'~'	126	T_TO	To command token	'to'	
's'	83	T_STEP	Step command token	'step'	
'N'	78	T_NEXT	Next command token	'next'	
'W'	87	T_WHILE	While command token	('do while' | 'dowhile' | 'while')	
'L'	76	T_LOOP	Loop command token	('loop' | 'wend' | 'end while')	
'w'	119	T_EXITWHILE	Exit While command token	('exit do' | 'exit while')	
'f'	101	T_EXITFOR	Exit For command token	'exit for'	
'H'	72	T_HALT	Halt / end command token	('halt' | 'end')	
'D'	68	T_DEFSUB	Sub definition token	'sub'	
'd'	100	T_ENDSUB	End sub token	('end sub' | 'endsub')	
'x'	120	T_EXITSUB	Exit sub token	('exit sub' | 'exitsub')	
'C'	67	T_CALL	Call command token	'call'	
'U'	85	T_DEFFUNC	Function definition token	'function'	
'u'	117	T_ENDFUNC	End function token	('end function' | 'endfunction')	
'R'	82	T_EXITFUNC	Exit function token	('exit function' | 'exitfunction')	
'M'	100	T_DIM	Dim command token	'dim'	
'a'	97	T_AS	As command token	'as'	
'0'	48	T_TYPE_INT	Integer type specifier	'integer'	
'"'	34	T_CONST_STRING	String Constant	('"' ?* '"')	1
'#'	35	T_CONST_INT	Integer Constant	([0-9]+)	2
'V'	86	T_VARIABLE	Variable Token	([a-z] ([a-z]|[0-9]|'_')*)	3
'B'	46	T_BUILTIN	Built-in specifier token	Matches a built-in name.	4
'“'	39	T_COMMENT	Comment Specifier	('“'| '#' | 'rem') (?*) (\r|\n)	5

Token Notes:

1. T_CONST_STRING (string constant) tokens are stored as a string of characters enclosed within double quotes. For example: “this is a string token”
2. T_CONST_INT (integer constant) tokens are 3 bytes stored as #[MSB][LSB] where MSB and LSB combine to form a signed integer.
3. T_VARIABLE (variable specifier) tokens are 2 bytes stored as V[VARIABLE_NUM] where [VARIABLE_NUM] represents a unique numeric identifier for the variable name.
4. T_BUILTIN (built-in specifier) tokens are 2 bytes stored as B[BUILTIN_NUM] where [BUILTIN_NUM] represents the number index of the builtin functions as follows: 0 = Pass, 1=SetDIOHigh, 2=SetDIOLow, 3=SetDIODirectionIn, 4=SetDIODirectionOut, 5=ReadDIO, 6=ReadAD, 7=ServoEnable, 8=ServoDisable, 9= ServoDisabledStateHigh, 10=ServoDisabledStateLow, 11=QuickMoveServoScaled, 12=QuickMoveServoPercent, 13=MoveServoScaled, 14=MoveServoPercent, 15=TimedMoveServoScaled, 16=TimedMoveServoPercent, 17=QuickLoadPreset, 18=CrossfadePreset,19=delaySec, 20=delayMilliSec, 21=Srand, 22=Rand, 23=RandRange, 24=kbhit, 25=getch, 26=putch, 27=CmdAr, 28=TimeMSec, 29=TimeSec, 30=TimeMin, 31=TimeClear, 32=SetLedMode.
5. T_COMMENT (comment) tokens are stored as a string of characters enclosed within single quotes. For example: 'This is a comment token'

6.4 Error Codes

When an executing SC-BASIC sequencer program encounters an error, the execution is halted and the error code byte is set to indicate the error encountered. The following list provides a description of each error code.

0 = “No Error"
1 = "ERROR: assignment or function call expected after EQUAL."
2 = "ERROR: NEXT without FOR."
3 = "ERROR: LOOP without WHILE."
4 = "ERROR: ELSE without IF."
5 = "ERROR: ELSEIF without IF."
6 = "ERROR: ENDIF without IF."
7 = "ERROR: FUNCTION/SUB cannot be defined inside SUB."
8 = "ERROR: FUNCTION/SUB cannot be definied inside FUNCTION."
9 = "ERROR: BUILTIN or VARIABLE expected after CALL."
10 = "ERROR: ENDSUB without SUB."
11 = "ERROR: EXITSUB without SUB."
12 = "ERROR: ENDFUNC without FUNCTION."
13 = "ERROR: EXITFUNC without FUNCTION."
14 = "ERROR: unexpected token in statement."
15 = "ERROR: VARIABLE expected in DIM."
16 = "ERROR: Unsupported type STRING in DIM."
17 = "ERROR: Unsupported type SINGLE in DIM."
18 = "ERROR: Unexpected token after AS in DIM."
19 = "ERROR: unexpected token in DIM."
20 = "ERROR: variable name expected after SUB/FUNCTION call."
21 = "ERROR: Missing right parenthesis on SUB/FUNCTION parameter list."
22 = "ERROR: Unexpected token in parameter list of CALL."
23 = "ERROR: unexpected token in CALL."
24 = "ERROR: Unexpected token in SUB parameter list."
25 = "ERROR: Missing right parenthesis on parameter list in SUB definition."
26 = "ERROR: Unexpected token in SUB parameter list."
27 = "ERROR: Unsupported return type STRING in FUNCTION definition."
28 = "ERROR: Unsupported return type SINGLE in FUNCTION definition."
29 = "ERROR: Unexpected token after AS in FUNCTION definition."
30 = "ERROR: Unexpected token in SUB."
31 = "ERROR: parameter count mismatch in call."
32 = "ERROR: Unexpected token in WHILE loop."
33 = "ERROR: EOF encountered within WHILE."
34 = "ERROR: variable expected after 'FOR'."
35 = "ERROR: EQUAL expected after variable in FOR."
36 = "ERROR: missing TO in FOR statement."
37 = "ERROR: unexpected token after FOR."
38 = "ERROR: variable mismatch in NEXT."
39 = "ERROR: Unexpected token in FOR loop."
40 = "ERROR: EOF encountered in FOR."
41 = "ERROR: Missing THEN in IF statement."
42 = "ERROR: EOF encountered within IF."
43 = "ERROR: unexpected token in ELSE clause."
44 = "ERROR: Missing THEN in ELSEIF statement."
45 = "ERROR: unexpected token following ELSEIF"
46 = "ERROR: unexpected token in IF."
47 = "ERROR: Unexpected Token found - Expression expected."
48 = "ERROR: Missing right-parenthesis in expression."
49 = "ERROR: unexpected token in expression."
50 = "ERROR: variable expected as lvalue in assignment."
51 = "ERROR: invalid assignment, EQUAL expected."
52 = "ERROR: invalid token in PRINT."
53 = "ERROR: invalid builtin call, (expected."
54 = "ERROR: invalid builtin call,) expected."
55 = "ERROR: unknown built-in function."
56 = "ERROR: division by zero."
57 = "ERROR: DSTACK overflow."
58 = "ERROR: HSTACK overflow."
59 = "ERROR: Improperly formed CONST_STRING."
60 = "ERROR: Improperly formed COMMENT."
61~99 = Reserved

100 = "ERROR: CALL_STACK overflow."
101 = "ERROR: CALL_STACK underflow."
102 = "ERROR: Duplicate SUB/FUNCTION definition."
103 = "ERROR: FUNCTION/SUB table full."
104 = "ERROR: an identifier VARIABLE must follow a SUB definition."
105 = "ERROR: missing end parenthesis in SUB declaration."
106 = "ERROR: unexpected token in SUB parameter list."
107 = "ERROR: unexpected token in SUB declaration."
108 = "ERROR: EOF encountered before ENDSUB."
109 = "ERROR: SUB definition encountered within SUB."
110 = "ERROR: FUNCTION definition encountered within SUB."
111 = "ERROR: EXITFUNCTION encountered within SUB."
112 = "ERROR: ENDFUNCTION encountered within SUB."
113 = "ERROR: an identifier VARIABLE must follow a FUNCTION definition."
114 = "ERROR: missing end parenthesis in FUNCTION declaration."
115 = "ERROR: unexpected token in FUNCTION parameter list"
116 = "ERROR: unexpected token in FUNCTION declaration."
117 = "ERROR: EOF encountered before ENDFUNCTION"
118 = "ERROR: SUB definition encountered within FUNCTION."
119 = "ERROR: FUNCTION definition encountered within FUNCTION."
120 = "ERROR: EXITFUNCTION encountered within FUNCTION."
121 = "ERROR: ENDFUNCTION encountered within FUNCTION."
122 = "ERROR: Unsupported type STRING in parameter list"
123 = "ERROR: Unsupported type SINGLE in parameter list"
124 = "ERROR: Unexpected token after AS in parameter list"
125 = "ERROR: variable not found."
126 = "ERROR: variable table full."
127 = "ERROR: SUB/FUNCTION not defined."